MSBD 5001 Midterm

Pranav A, 666666

December 5, 2017

1 Supervised Learning

For the classification methods, I have set up the following pipeline.

1. Scaling the features to zero mean and unit variance.
2. Since it is a high dimensional data and the given time is limited for this exam, PCA is applied with just 2 components.
3. Application of choice classifer on the resulting dataset.

The dataset is split according to the ratio of 80:20 of training against test set. My choice classifiers are (collection of classifiers from the scikit-learn library):

1. **Logistic Regression**: Standard logistic regression defines the linear combination of weights of the features. This definition is embedded further into the negative log likelihood function. The goal of this classifier is to maximize this log likelihood function. Here the parameters I have setup are L_2 penalization, tolerance of 1×10^{-4} and regularization parameter $C = 1$.
2. **Bayesian Ridge Regression**: Here the likelihood function would be the Gaussian inferences of the linear regression based weights (including the bias). Since we have the cross-validation dataset, the hyperparameters would be tuned over that dataset. The tolerance I have taken is 0.001 and the number of iterations are 300.
3. **Lasso**: This is the linear regression function, but with L1 prior as regularizer. Parameters I have taken are tolerance of 0.0001 and $\alpha = 0.1$ and rest are the default parameters.
4. **Support Vector Classifier**: This maximizes the margins between the positive and negative classes. Here I have implemented the Gaussian kernel with $C = 1$ and the tolerance of 0.0001.
5. **Linear classifer with SGD**: This is the linear classifier. Only difference is the I have used Stochastic Gradient Descent here for minibatching the gradient losses. Another way to look at this as the linear SVM. Default parameters are used here.

Classifiers	Validation Accuracy	Test set accuracy
Logistic Regression	0.949	0.947
Bayesian Ridge	0.659	0.616
Lasso	0.648	0.59
SVC	0.938	**0.956**
SGD Classifier	**0.952**	0.93

Table 1: Accuracies for the given classifiers

For the cross-validation hyperparameter training, I have done a 5-fold validation with stratified sampling. After training the hyperparameters, the average of accuracies have been reported in Table 1.

The classification accuracy table for the test data set is shown in Table 1. I observe that SVC is performing the best. This is due to Gaussian kernels giving complex boundaries resulting in a higher accuracy. The linear classifiers like logistic regression and SGD trained linear model also perform well here. The key here of performing well is due to PCA decomposition as a preprocessing. Otherwise, they would not have performed well. Lasso and Bayesian Ridge did not perform here well because the assumptions carried by these models were too constrained on the data.

2 K-means clustering

2.1 Definition of explained variance of k-means and elbow method

For clustering, elbow method would be used for the given algorithm. The quality of the k-means can be given by the explained variance of the given clusters.

I define x_{ij} as the given cluster point for the i^{th} observation for the given j^{th} feature. Let $d_{ii'}$ be the distance calculated within the observations i and i', which is usually defined by the euclidean distance. Let the data be clustered according to the k clusters, with centroids given as $C_1, C_2, \ldots, C_k$ with C_x denoting the number of points contained in the cluster x. Let the number of points be $n_x = |C_x|$.

Then D_x be the sum of the pairwise distances in the given dataset in the given cluster x.

Thus, we have:

$$D_x = \sum_{i,i' \in C_x} d_{ii'}$$

Then I would define the W_k as the combination of within-sum distances of the clusters. Hence, W_k would be defined as,

$$W_k = \sum_{x=1}^{k} \frac{1}{2n_x} D_x$$

Elbow method uses the sudden change of the discontinuity of the loss of W_k. This sudden change of the discontinuity of the loss can be visualized as the maximum slope difference between the W_{k-1} and W_{k+1}.

Slope of W_k is

$$slope_{W_k} = tan^{-1}\left(\frac{|W_{k-1} - W_{k+1}|}{(k+1)-(k-1)}\right)$$

Thus the optimal k, that is k^* is

$$k^* = \arg\min_k tan^{-1}\left(\frac{|W_{k-1} - W_{k+1}|}{2}\right)$$

2.2 Dataset application

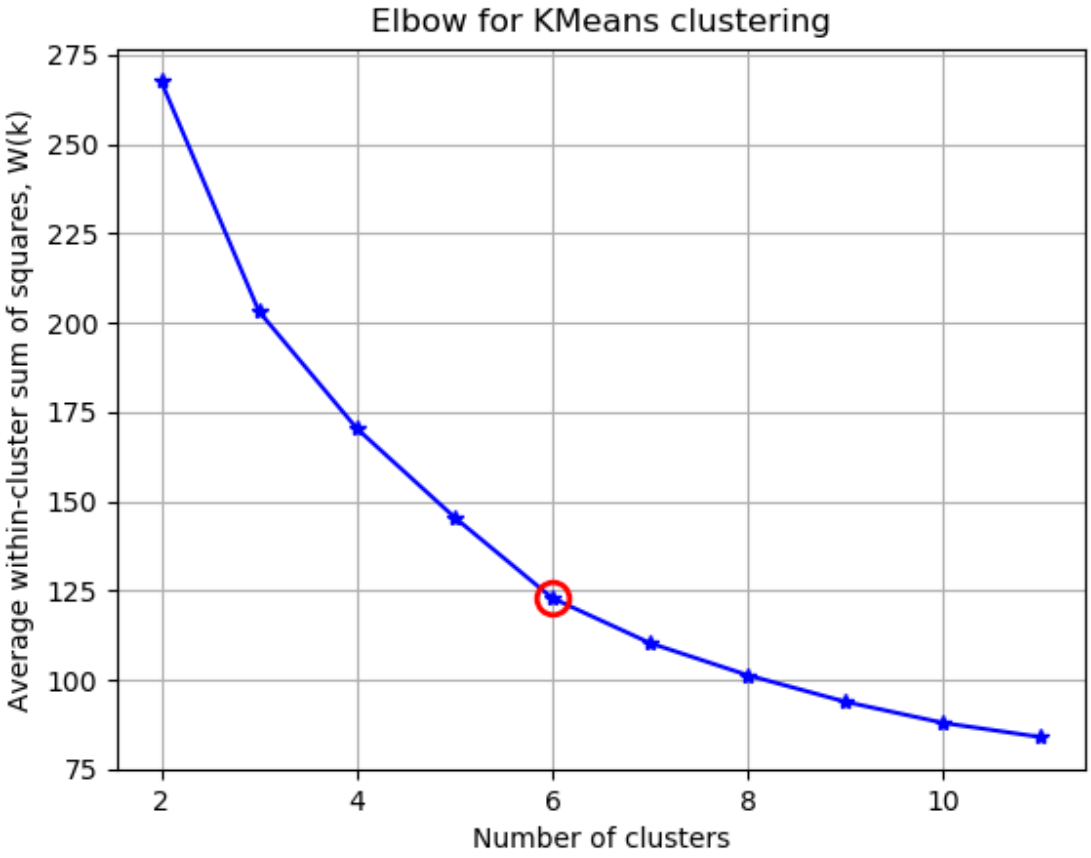

Figure 1: Elbow method of the given clusters for the breast cancer dataset

For the given dataset, I have initially made the data to be of zero mean and unit variance, after dropping the class attribute. After that I have used k-means clustering and calculated the residual errors as described above from $k = 2$ till 11. The plot of the within-sum clusters corresponding to the number of clusters have been plotted in the figure 1. After that I calculated maximum slope difference for the given nearby clusters and found out $k = 6$ being the optimal number of clusters.

3 Skewed Datasets

Literature in Machine Learning mainly suggest three approaches to classification algorithms for skewed datasets.

1. Oversampling the minority dataset
2. Undersampling the majority dataset
3. Skew-sensitive learning

I will briefly describe how these three processes would work for the given classification algorithms and also suggest some tweaks that could work for these classifiers.

3.1 Decision trees

Instead of taking information gain as a splitting criteria, one could take skewness as the splitting criteria. For that, one could use Hellinger Distance to measure the divergence of skewed classes.

The distance between positive X_p and negative class X_n for Hellinger Distance is given by,

$$d(X_p, X_n) = \sqrt{\sum_{j=1}^{p} \left(\frac{|X_{pj}|}{|X_p|} - \frac{|X_{nj}|}{|X_n|} \right)^2}$$

where the $|X_{pj}|$ just means the length of the vector j belonging to the positive class.

Whichever feature is responsible for the highest skew, that means maximum of Hellinger Distance, that would be our splitting criterion. Thus I will take the arg-max of the distances and split at the point. The child node will be created for that feature which I need to split. One could run decision tree recursively, till the desired cut-off has been reached.

In my opinion, the oversampling and undersampling should happen in conjunction for decision tree, as it is a hard classifer.

The research literature shows that the combination of random undersampling and the oversampling often leads to a better performance.

Although it has been shown that conjunction of skew-sensitive learning (like HDDT) with sampling wrapper don't give a significantly better performance.

3.2 Adaboost

3.2.1 Oversampling techniques

In Adaboost, one could generate new dataset examples by following the positive class' feature distribution. This can be done using many advanced dataset augmentation techniques or just simply generation based preprocessing techniques like SMOTE or MSMOTE. One can do this dataset augmentation before the estimation of error rate. One has to make sure to keep the ratio of weights between the generated samples against the original samples. This way,

the weightage of the generated samples would be the same, and original samples' weights would be renormalized and reupdated. The augmentation phase can be applied again to the dataset after the initial training to avoid overfitting.

3.2.2 Undersampling techniques

In the similar fashion, one can remove samples from the majority class before the training of the individual classifiers. In this way, the weights do not need to be normalized again. Rest of the procedure is same as the oversampling one.

3.2.3 Skew measure based techniques

I introduce here, the penalization function as the function of skew between the datasets. If the skewness is measured by b_1, then the penalization function for the classifer i to be learned is $p_i = f_i(b_1)$ for the given misclassification.

The update equation of the dataset distribution is now given by,

$$D_{t+1}(i) = p_i D_t(i) e^{-w_t h_t(x) y}$$

The weight of classifers would be,

$$w_t = \frac{1}{2} \ln \frac{\sum_{i, y = h_t(x_i)} p_i D_t(i)}{\sum_{i, y \neq h_t(x_i)} p_i D_t(i)}$$

Here the p_i can be thought of as the weight for the classifiers' dataset distribution.

3.3 Naive Bayes Algorithm

Mainly literature suggests one-vs-all classification, however this would not work here as it is a binary classification. I propose three tweaks here which could help naive bayes classification for the skewed classes.

3.3.1 Smoothing

Smoothing of the parameters in the negative log likelihood of the naive bayes classifier, would really help avoid getting extreme probabilities.

In my view, Dirichlet smoothing really helps without distorting the actual probabilities too much.

However, just this tweak should not be used alone for naive bayes.

3.3.2 Sampling

One can create the small block samples datasets, where in each block, the classes are nearly balanced. If uniform distribution is observed for each block, the resulting classifiers would be the average over all these blocks.

The balanced dataset in each block can be created by randomly undersampling the majority class and oversampling (generating synthetic data) the minority classes.

3.3.3 Thresholding

For heavily skewed class, we would learn the approximate probabilities by learning thresholds of minority class only. Here we are leaving out the majority class for the classification.

That means,

$$t_i = min_x \left(\prod_{j=1}^{n} p(A_j = v_i) \right)$$

The above equation sets the threshold t_i for each feature i. This threshold the minimum product for the training example x where the attribute A_j for the value v_i.

Since we set the minimum value from the possible training examples, that means any value exceeding this threshold belongs the positive class.

Mathematically, we would get the positive class if in the testing set we have $\prod_{j=1}^{n} p(A_j = v_i) \geq t_i$.

4 Markov Decision Process

In this MDP problem, I need to give a rigid definitions of the transition models and rewards. Once these are adequately defined, I can plug these values in the value iteration algorithm.

4.1 Transition Model

In order to minimize the total expected cost, we need to find the probability distribution and the rewards for the states.

Let s be the set of cities from $i = 1$ to n. The probability of moving from city successfully s_i to s_j is given by $T(s_i, s_j, a_{ij})$. The probability of having that path blocked is $1 - T(s_i, s_j, a_{ij})$. The probability of staying in the city s_i would only happen if all roads from city s_i to distinct cities s_j are blocked. Assuming the conditional independence that roads are blocked independently of each other. The probability the agent has to stay in the city is given by, $T(s_i, s_i, a_{ii}) = \prod_{k \in (S - S_i)} (1 - T(s_i, s_k, a_{ik}))$

Now I define the probability distribution over the city s_i. I noticed that the emission probabilities from the city s_i do not sum up to 1. Hence, it's essential to normalize these transition probabilities for a well-defined probability distribution.

Hence the normalization factor constraint (c_i) for the city s_i would be just the sum of all these transition probabilities.

$$\left(\prod_{k \in S - S_i} (1 - T(s_i, s_k, a_{ik})) \right) + \sum_{k \in S - S_i} T(s_i, s_k, a_{ik}) = c_i$$

Thus the probability from going city s_i to s_j is,

$$P(s_j | s_i, a) = \frac{T(s_i, s_j, a_{ij})}{c_i} = \frac{T(s_i, s_j, a_{ij})}{\left(\prod_{k \in S - S_i} (1 - T(s_i, s_k, a_{ik})) \right) + \sum_{k \in S - S_i} T(s_i, s_k, a_{ik})}$$

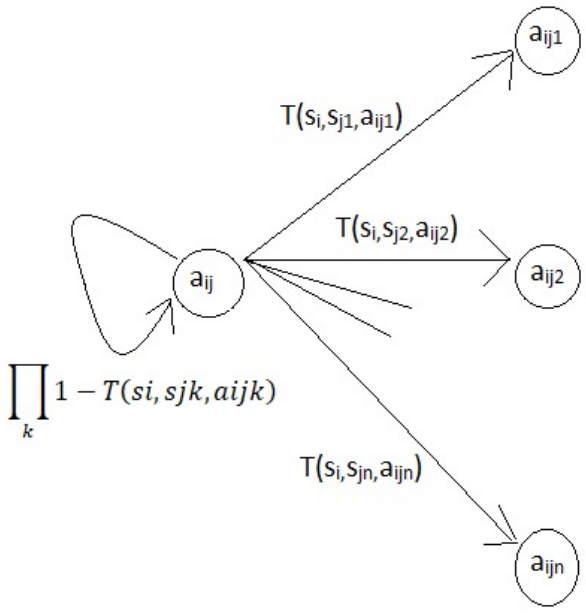

Figure 2: Transition probabilities of the given actions

And probability of staying in city s_i would be,

$$P(s_i|s_i,a) = \frac{T(s_i,s_i,a_{ii})}{c_i} = \frac{\prod_{k\in S-S_i}(1-T(s_i,s_k,a_{ik}))}{\left(\prod_{k\in S-S_i}(1-T(s_i,s_k,a_{ik}))\right)+\sum_{k\in S-S_i}T(s_i,s_k,a_{ik})}$$

Thus the transition model looks like,

$$P(s_j|s_i,a) = \begin{cases} \frac{T(s_i,s_j,a_{ij})}{c_i} = \frac{T(s_i,s_j,a_{ij})}{\left(\prod_{k\in S-S_i}(1-T(s_i,s_k,a_{ik}))\right)+\sum_{k\in S-S_i}T(s_i,s_k,a_{ik})} & \text{if } s_j \neq s_i \\ \frac{T(s_i,s_i,a_{ii})}{c_i} = \frac{\prod_{k\in S-S_i}(1-T(s_i,s_k,a_{ik}))}{\left(\prod_{k\in S-S_i}(1-T(s_i,s_k,a_{ik}))\right)+\sum_{k\in S-S_i}T(s_i,s_k,a_{ik})} & \text{if } s_j = s_i \end{cases} \quad (1)$$

4.2 Reward function

The given reward would depend on how the transition happened from the time period t from $t-1$.

The given task wants to minimize the cost. However, in the value iteration algorithm, we usually maximize the rewards. This is analogous to maximizing the negative cost.

Two different reward function matrices will be formed according to the problem. If the transition happened between two states for the time period t from $t-1$ from s_i to s_j, the reward function would be $R(s) = -R(s_i,s_j)$. If the transition doesn't happen between two states for the time period t from $t-1$, the reward function would be $R(s) = -R(s_i,s_i)$.

In the nutshell, the reward function is,

$$R(s) = \begin{cases} -R(s_i,s_j) & \text{if } s_j^t \neq s_i^{t-1} \\ -R(s_i,s_i) & \text{if } s_j^t = s_i^{t-1} \end{cases} \quad (2)$$

4.3 Value Iteration algorithm

Plugging these values in value-iteration formula, we get

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

This is the value-iteration algorithm as given in the Russell Norvig's chapter. Here $R(s)$ is the reward function as defined by the equation 2 and the transition model $P(s'|s,a)$ is given by the equation 1. The $U(s)$ is the vector of utilities which would be iteratively updated and the discount factor is γ.

Notes

1. This given solution of the midterm follows the HKUST honour code. The solutions haven't been discussed with anyone and solutions provided are my own.

2. Code and references are omitted in this document as the library commands of the code could result into higher similarity index in TurnItIn. They are available on the request in the form of Jupyter Notebooks.